SCENARIO

The application a serialization-based session mechanism and a certain feature invokes a dangerous method on data provided in a serialized object. We will try to edit the serialized object in the session cookie to exploit this vulnerability and and use it to delete the morale.txt file.

**PROCEDURE**

1. Go the application and login using the given credentials to act as an user.
2. Navigate to the **My Account** page and inspect the source to see that we need to hit an URL endpoint to delete the user.
3. Open the BurpSuite’s Proxy’s HTTP History and study the request carefully.
4. We notice that the session cookie is encoded in Base64 and then in URL encoding.
5. We decode it in BurpSuite and we can see that it appears in the format below:

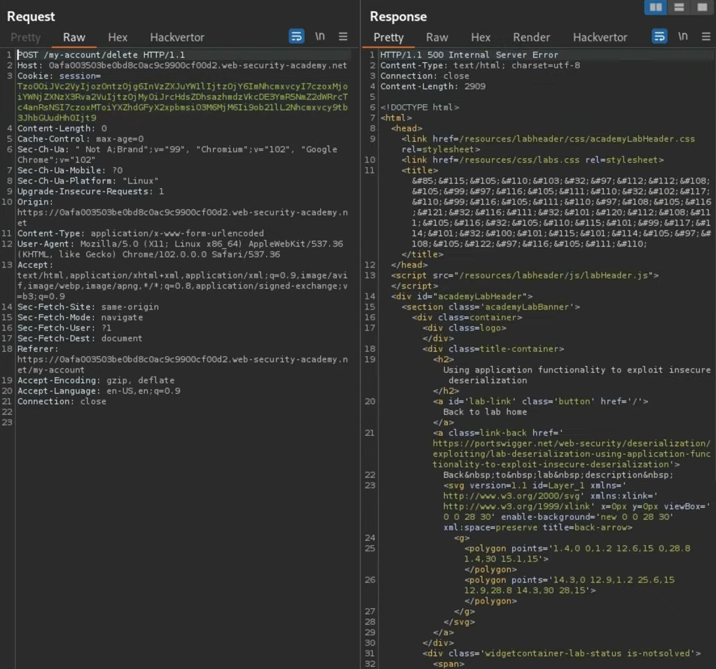
**O:4:"User":3:{s:8:"username";s:6:"wiener";s:12:"access\_token";s:32:"op9eukqyolydfc1uf5emvf0s39bc3m0c";s:11:"avatar\_link";s:19:"users/wiener/avatar";}**

1. So, we inject the payload in place of the above value.
2. Now we access the **Admin Panel** using the response we got and delete the user by hitting the endpoint we found in the page.

**PAYLOAD**

O:4:"User":3:{s:8:"username";s:6:"wiener";s:12:"access\_token";s:32:"op9eukqyolydfc1uf5emvf0s39bc3m0c";s:11:"avatar\_link";s:23:"/home/carlos/morale.txt";}

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **Avoid Unsafe Deserialization:** Unsafe deserialization can lead to a variety of attacks. Use libraries that allow for safe deserialization and avoid those that can deserialize into live objects or execute methods. If the application does not need to deserialize objects, then this feature should be turned off.
2. **Implement Input Validation:** Ensure that serialized objects are well-formed before processing them. Apply strict type constraints during deserialization to prevent the creation of unexpected object types.
3. **Signed Serialized Objects:** If serialized data needs to be sent to the client-side, make sure that it's digitally signed. By checking the signature before deserializing, you can ensure the data has not been tampered with.
4. **Use a Safe Serialization Format:** Consider using safer formats for serialization like JSON, which don't support the execution of methods. If you're using Java, consider moving from native serialization to formats like Protocol Buffers or BSON.
5. **Limit the Attack Surface:** If a certain functionality requires the use of serialization, restrict it to only the necessary parts of the application and avoid exposing it broadly.